QoS-Driven Selection of Composable Web Services

Zeina Azmeh, Driss Maha, Marianne Huchard, Moha Naouel, Chouki Tibermacine

QoS-Driven Selection of Composable Web Services

Zeina Azmeh[1], Maha Driss[2,3],
Marianne Huchard[1], Naouel Moha[2], and Chouki Tibermacine[1]

[1] LIRMM, CNRS and Montpellier University, France
{azmeh,huchard,tibermacin}@lirmm.fr
[2] IRISA/INRIA,University of Rennes I, France
{mdriss,moha}@irisa.fr
[3] RIADI-GDL Laboratory, University of Manouba, Tunisia

Abstract. Web Services are universally accessible software units that are advertised, discovered, and invoked over the internet. As web service technology is becoming widely adopted by organizations that need to integrate their information systems within and across organizational boundaries, it is important to identify automatically relevant composable services. In this paper, we propose an approach based on a variant of Formal Concept Analysis for the identification of composable services. It allows the classification of web services according to their QoS and composability levels and proposes a query mechanism that allows users to specify their required QoS and composability levels. In the case of multiple choices that satisfy users' requirements, we enhance our approach with a mechanism based on vectors for identifying the optimized choice. We validate our approach on a set of real-world web services obtained from Service-Finder.

1 Introduction

Service-Oriented Computing (SOC) is an emerging paradigm for developing low-cost, flexible, and scalable distributed applications based on web services [10]. Web services are autonomous, reusable and independent software units that can be accessed through the internet. SOC is becoming broadly adopted and in particular by companies, which are more and more willing to open their information systems to their clients and partners over the Internet. The reason comes from the fact that SOC offers the ability to build efficiently and effectively added-value service-based applications by composing ready-made services. Web service composition addresses the situation when the functionality required by users cannot be satisfied by any available web service, but by assembling suitably existing web services [18]. Discovering and selecting relevant services that closely fit users' functional and non-functional requirements is an important issue highly studied in the literature [24,1,23]. Functional requirements define functionalities provided by web services and non-functional requirements define Quality of Service (QoS) criteria such as availability, response time, security, and throughput, etc. [14]. However, discovering and selecting relevant composable services is

another important issue that still need to be investigated, since few approches focused on service composability problem [12,7,11]. Relevant composable services represent services minimize the amount of adaptation among them while best fitting requirements.

In this paper, we propose an approach for the identification of composable services that best fit QoS and adaptation requirements. This approach is based on the formal concept analysis (FCA) and its variant, the relational concept analysis (RCA). FCA has been successfully applied in the past as a formal framework for service substitution [16,2,3,4,6]. Thus, we use FCA and RCA to classify services according to their QoS and composability level and suggest substitutable services and composable services. The approach is also based on a query mechanism that allows users to specify their required QoS and composability levels. In particular, users can specify their interests for identifying composable services that maximize QoS requirements and minimize their adaptation level. In the case of multiple choices that satisfy users' requirements, we enhance our approach with a mechanism based on vectors for identifying the optimized choice.

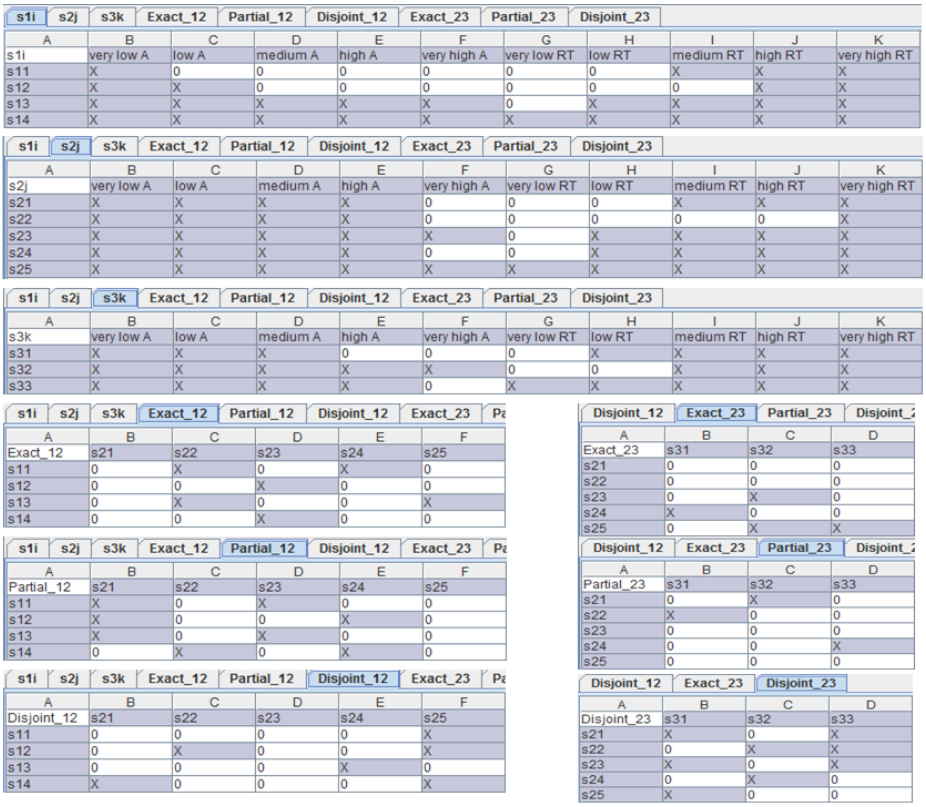
The paper is organized as follows. Section 2 describes our approach along with a motivating example. Section 3 describes the experiments performed on a real case study for validating our approach. The paper ends with the related work in Section 4 and the conclusion in Section 5.

2 Approach

We explain our approach along with an abstract orchestration of three sequential tasks. We try to identify the concrete services offering the needed functionalities, in order to instantiate the defined orchestration. Our objective is to select composable services with the highest affordable QoS values, and the least required adaptations. We have 3 sets of services: S_{1i}, S_{2j} and S_{3k} corresponding to the three tasks. For each service of the three sets, we assume certain values of QoS (for availability (A) and response time (RT)). We categorize the QoS values into 5 levels: very low, low, medium, high and very high, for each of the A and RT values. A service that offers a specific QoS level, does also offer QoS levels that are lower. For each pair of consecutive services (according to our defined orchestration), we assume three composability values: exact, partial and disjoint.

RCA-Based Classification. We base our classification on Formal Concept Analysis (FCA, [8]), which is a data analysis method that aims at extracting concepts from entities described by attributes. The description of entities is encoded in a formal context. For example, in Fig. 1, the third table S_{3k} is a formal context where entities are services for the Task 3, and are described by the values of availability and response time (attributes). Concepts are maximal sets of entities (extent) sharing a maximal set of attributes (intent), and they are organized in a classification provided with a lattice structure. The lattice associated with the context S_{3k} is shown in Fig. 5. Each concept in this lattice inherits the attributes of its ascendants (super-concepts), and has lesser attributes than its descendants (sub-concepts). Reversely, each concept inherits the services of its

descendants (sub-concepts). For example, the concept c_0 represents the services s_{31} and s_{33}, which covers the QoS values of the services in the upper concepts, and offers better ones (like: low A and low RT). The concept c_5 indicates that s_{33} has very low RT and high A. The concept c_5 is a sub-concept of the concept c_0, which shows that s_{33} has better QoS values than s_{31}.



s1i | s2j | s3k | Exact_12 | Partial_12 | Disjoint_12 | Exact_23 | Partial_23 | Disjoint_23

A	B	C	D	E	F	G	H	I	J	K
s1i	very low A	low A	medium A	high A	very high A	very low RT	low RT	medium RT	high RT	very high RT
s11	X	0	0	0	0	0	0	X	X	X
s12	X	X	0	0	0	0	0	0	X	X
s13	X	X	X	X	X	0	X	X	X	X
s14	X	X	X	X	X	X	X	X	X	X

s1i | s2j | s3k | Exact_12 | Partial_12 | Disjoint_12 | Exact_23 | Partial_23 | Disjoint_23

A	B	C	D	E	F	G	H	I	J	K
s2j	very low A	low A	medium A	high A	very high A	very low RT	low RT	medium RT	high RT	very high RT
s21	X	X	X	X	0	0	0	X	X	X
s22	X	X	X	X	0	0	0	0	0	X
s23	X	X	X	X	X	0	X	X	X	X
s24	X	X	X	X	0	0	X	X	X	X
s25	X	X	X	X	X	X	X	X	X	X

s1i | s2j | s3k | Exact_12 | Partial_12 | Disjoint_12 | Exact_23 | Partial_23 | Disjoint_23

A	B	C	D	E	F	G	H	I	J	K
s3k	very low A	low A	medium A	high A	very high A	very low RT	low RT	medium RT	high RT	very high RT
s31	X	X	X	0	0	0	X	X	X	X
s32	X	X	X	X	X	X	0	X	X	X
s33	X	X	X	X	0	X	X	X	X	X

s1i | s2j | s3k | Exact_12 | Partial_12 | Disjoint_12 | Exact_23 | Pa

A	B	C	D	E	F
Exact_12	s21	s22	s23	s24	s25
s11	0	X	0	X	0
s12	0	0	X	0	0
s13	0	X	0	0	X
s14	0	0	X	0	0

Disjoint_12 | Exact_23 | Partial_23 | Disjoint_

A	B	C	D
Exact_23	s31	s32	s33
s21	0	0	0
s22	0	0	0
s23	0	X	0
s24	X	0	0
s25	0	X	X

s1i | s2j | s3k | Exact_12 | Partial_12 | Disjoint_12 | Exact_23 | Pa

A	B	C	D	E	F
Partial_12	s21	s22	s23	s24	s25
s11	X	0	X	0	0
s12	X	0	0	X	0
s13	X	0	X	0	0
s14	0	X	0	X	0

Disjoint_12 | Exact_23 | Partial_23 | Disjoint_

A	B	C	D
Partial_23	s31	s32	s33
s21	0	X	0
s22	X	0	0
s23	0	0	0
s24	0	0	X
s25	0	0	0

s1i | s2j | s3k | Exact_12 | Partial_12 | Disjoint_12 | Exact_23 | Pa

A	B	C	D	E	F
Disjoint_12	s21	s22	s23	s24	s25
s11	0	0	0	0	X
s12	0	X	0	0	X
s13	0	0	0	X	0
s14	X	0	0	0	X

Disjoint_12 | Exact_23 | Disjoint_23

A	B	C	D
Disjoint_23	s31	s32	s33
s21	X	0	X
s22	0	X	X
s23	X	0	X
s24	0	X	0
s25	X	0	0

Fig. 1. The relational contexts family (RCF) corresponding to the three sets of services.

Since we also want to capture the composition relations, existing between pairs of services in an orchestration, we use a variant of FCA called Relational Concept Analysis (RCA), which includes relations in the attributes used for the concept mining [9]. RCA data are given in the form of a Relational Context Family (RCF). An RCF is a set of binary relationships (tables): some (non-relational) tables describe entities by attributes, while some (relational) tables describe relations between entities. The non-relational part of the RCF contains one table for each task: entities are services that can make the task, and attributes are QoS values. Thus, our RCF is composed of three non-relational contexts: $S_{1i} * QoS$, $S_{2j} * QoS$ and $S_{3k} * QoS$, and two sets of three relational contexts: composition of $S_{1i} * S_{2j}$ and composition of $S_{2j} * S_{3k}$, according to the three considered levels.

As we can see in Fig. 1, the three relational contexts for composition of $S_{xi} * S_{yj}$ are called $Exact_{xy}$, $Partial_{xy}$ and $Disjoint_{xy}$.

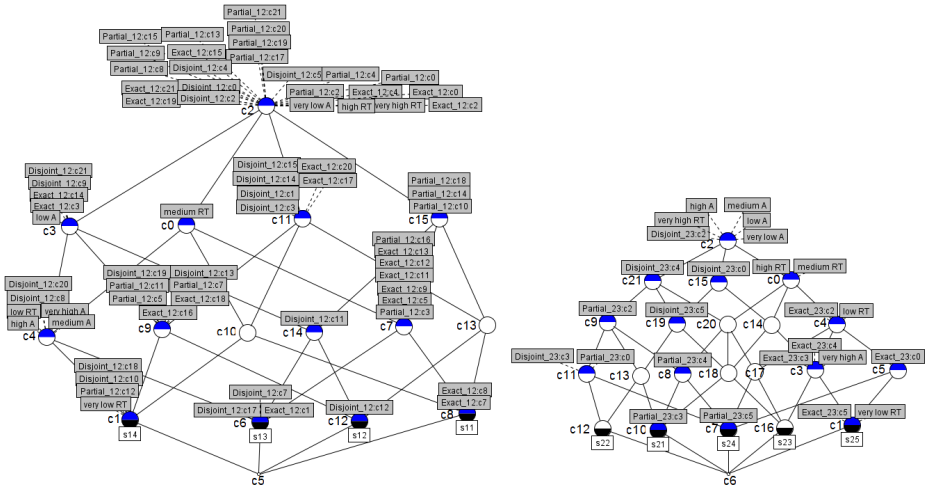


Fig. 2. The services lattices for Task 1 and Task 2.

We run the RCA algorithm on the RCF, illustrated in Fig. 1. The RCA takes as input all the tables and iterates on two steps: (1) building a concept lattice on each non-relational context of services concatenated with the corresponding relational contexts that express composition, that is on $S_{1i} * QoS$ concatenated with $Exact_{12}$, $Partial_{12}$ and $Disjoint_{12}$, on $S_{2j} * QoS$ concatenated with $Exact_{23}$, $Partial_{23}$ and $Disjoint_{23}$ and on $S_{3k} * QoS$; (2) transform the six relational contexts to integrate the concepts found at that current step (to use this knowledge in the next iteration). During this transformation, we use the existential scaling operator which is one of the scaling operators provided by RCA [9]. Services that form the columns of a composition context are replaced by concepts that group services. If (S_{xy}, S_{zt}) initially holds in a composition context (with composition level cl) and S_{zt} is in the extent of a concept C at the current step, thus in the current version of the composition context we add (S_{xy}, C). This can be interpreted as S_{xy} can be composed with at least one service from the set of services grouped in C with the composition level cl. For example, since we have $(s_{13}, s_{25}) \in Exact_{12}$ and since s_{25} is in the extent of Concept c_1 (in lattice S_{2j}), the scaled $Exact_{12}$ table will contain (s_{13}, c_1). By regarding the lattices in Fig. 2, we can find for example, that the service s_{13} (in the leftmost lattice) has in its intent $Exact_{12} : c1$. This means that it can be composed with services contained in the concept $c1$ (in the middle lattice). Thus, s_{13} can be composed at least with s_{25}. At a next step this will allow to group a set of services (implementing a first task) because they can be composed with services of another group of services (implementing another task). The process stops when no new concept emerge during the FCA analysis. At the end of the process, a concept lattice is generated for each set of services. In each lattice, services are classi-

fied into concepts, showing their QoS levels as well as their composability levels with other services, according to the orchestration. Concept c_7 in the S_{1i} lattice groups services s_{13} and s_{11}, and indicates that they can be partially composed with services of concept c_3 of the S_{2j} lattice, which groups s_{23} and s_{25}.

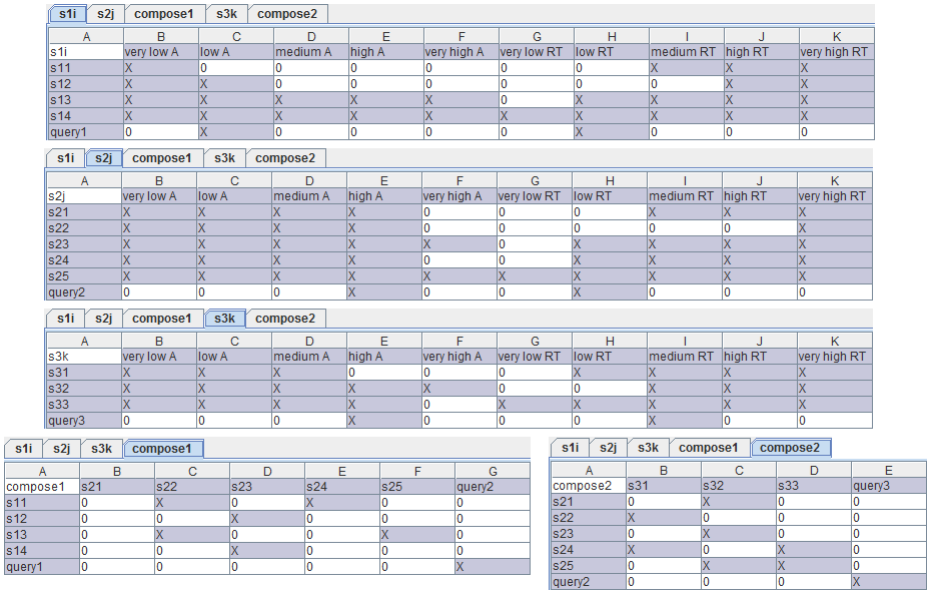


Fig. 3. The RCF after integrating the QoS queries and specifying the composability level.

Navigation by Query Integration. We define a query mechanism that enables us to navigate into a lattice, in order to choose a combination of services that satisfies the aimed orchestration, with our aimed levels of QoS and composability. A query determines the required QoS for each requested task, as well as, the needed composability level (required adaptations) between the services. We define a QoS query as a new line (a new entity), integrated into the corresponding service*QoS binary context, specifying the least expected QoS level. In our example, we perform three QoS queries: $query_1$, $query_2$ and $query_3$ on $S_{1i} * QoS$, $S_{2j} * QoS$ and $S_{3k} * QoS$ respectively. They are as follows: $query_1$ looks for a service from S_{1i}, with a low A and a low RT; $query_2$ tries to retrieve a service from S_{2j}, with a high A and a low RT; $query_3$ demands a service from S_{3k}, with a high A and a medium RT. In order to select services that satisfy a specific composability level, we merge the relational contexts that express levels equal or greater than the requested level. In other words, if a *partial* composability level can be accepted, this means that if we find an *exact* level, it is even better. In this example, we demand an "exact" composability level between S_{1i} and S_{2j}, and a "partial" level between S_{2j} and S_{3k}. This gives us a variation of our RCF that includes queries, where the relational context *compose*1 is the

context *exact*12 and *compose*2 is the fusion of *exact*23 and *partial*23, as shown in Fig. 3.

This latter RCF generates three lattices, from which we can extract the services combinations that meet our requested QoS and composability. These lattices are illustrated in Fig. 4, from which we can identify the services that satisfy the QoS queries, which appear in the sub-concepts of the concept where a $query_n$ appears, and are as follows: services s_{13} and s_{14} for $query_1$; services s_{23}, s_{24} and s_{25} for $query_2$; services s_{32} and s_{33} for $query_3$.

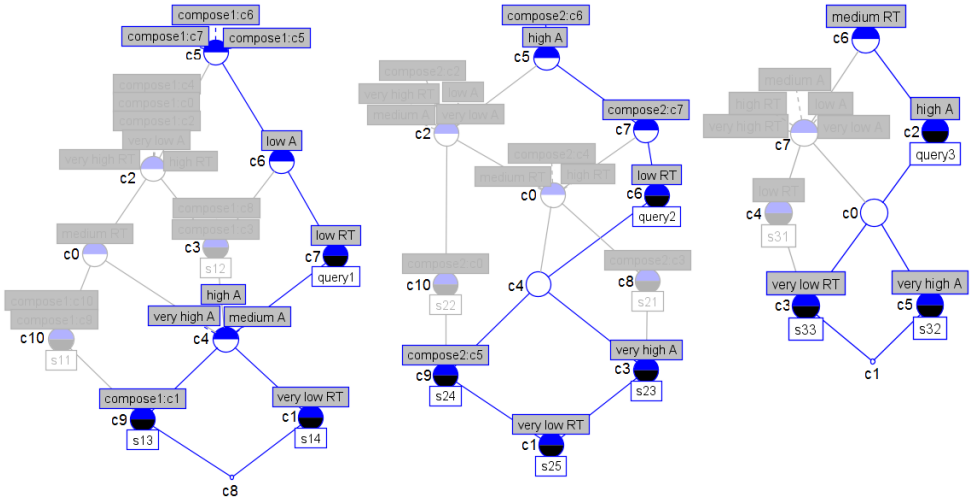


Fig. 4. The service lattices with the integrated queries.

The Optimal Services Composition. From the extracted services, we can have several combinations that satisfy our desired orchestration. If we are searching for an optimized selection, this means that we have to find the set of services that meet the optimal compromise of QoS levels and composability. Thus, services that can be coupled according to our acceptable levels of required adaptations and QoS. In order to meet this issue, we propose to represent the services by vectors. A vector per set of services, on which, services are ordered according to their QoS. We also define a vector for composability levels. Each composition can be regarded as a triangle, having a head corresponding to the composability level, and the two other heads corresponding to the pair of services to be composed, as shown in Fig. 6. The optimized choice would be the triangle that has the minimal area, in case of a two services composition. Otherwise, it will be the minimal sum of the services triangles according to an orchestration. Thus in our example, by regarding the triangles in Fig. 6, we notice that the triangle (E, s_{25}, s_{33}) represents an optimized composition between S_{2j} and S_{3k} (corresponding to Task2 and Task3, respectively). Accordingly, if we consider the triangle (E, s_{13}, s_{25}) that shares an edge with the previous triangle, it represents a composition between S_{1i} and S_{2j} (corresponding to Task1 and Task2, respectively).

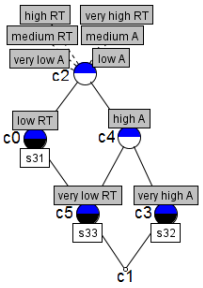


Fig. 5. The lattice associated with the context S_{3k} of Task 3.

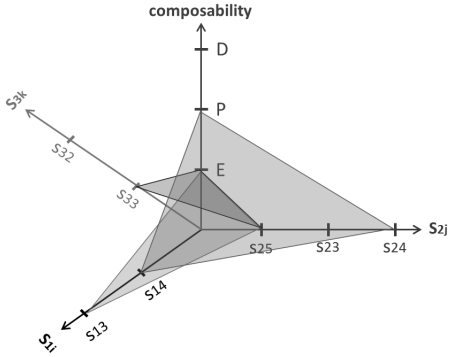


Fig. 6. Optimizing the service selection.

These two triangles together may be an optimized combination for the required orchestration, after comparing them with all the existing triangles[4].

3 Experiments

This section describes data and results of the experiments that we have conducted.

3.1 Experiments Data

To validate our approach, we consider an orchestration composed of three sequential tasks. The first task `IPToCity` returns the city name for a given IP address. The second task `CityToZipCode` returns the zip code for a given city name. The third task `ZipCodeToWeather` returns the weather information for a given zip code.
We discover a set of WSs that reply to each of these tasks by querying Service-Finder [22] [19]. Service-Finder is a Web 2.0 platform for WSs discovery that indexes almost 25.000 WSs. For example, we use keywords 'IP + City' to search WSs that achieve the first task. Service-Finder returns a result set of 36 WSs [5]. To reduce this set, we process three-step filtration. In the first step, we eliminate repeated WSs (i.e., WSs that offer exactly the same operations). This first filtration generates a new reduced set that contains 29 WSs. These WSs are passed to a second filtration step, in which we check whether the endpoint URI exists or not. This produces a new reduced set that contains 21 WSs. These WSs are passed to a third filtration step which leaves only services offering operations that satisfy the `IPToCity` task. This is done by parsing every service WSDL in order to check if the required operation exists or not. Finally, we obtain a set of

[4] We haven't shown all the triangles in Fig. 6, for the sake of simplicity.
[5] The discovery result set is obtained on June 18^{th}, 2010.

5 WSs. Table 1 gives for the three tasks, the keywords used to query Service-Finder, the number of WSs returned by it, and the number of the obtained WSs after each filtration step. The total number of WSs that reply to the three tasks is 21 WSs[6].

Tasks	Keywords used to query Service-Finder	# WSs returned by Service-Finder	# WSs obtained by the 1^{st} step filtration	# WSs obtained by the 2^{nd} step filtration	# WSs obtained by the 3^{rd} step filtration
`IPToCity`	'IP + city'	36	29	21	5
`CityToZipCode`	'City + Zip + Code'	166	140	112	12
`ZipCodeToWeather`	'Zip + Code + Weather'	9	9	7	4
				Total	**21**

Table 1. Summary of the obtained sets of services.

Table 2 is a synopsis of the 21 WSs used in our experiments, their operations, inputs, and outputs. For the sake of simplicity, we use the abbreviated notation 'S11' to refer to the service '1' that replies to the task '1' and 'Op11a' to refer to the operation 'a' from the service '1' that replies to the task '1', and so on. We consider 2 QoS properties to characterize WSs: availability (A) and response time (RT). Real time monitoring information of service availability and response time are provided by Service-Finder. To spread out availability and response time values, we use the boxplot statistical technique [5].

Table 3 gives availability and response time values for the 21 WSs.
To characterize the composability between two WSs, for example S11 and S21, we compute the semantic and structural similarity between the output and input parameters of their operations. We obtained values (Sim) $\in$ [0,1]. Then, we used a standard reasoning inferences [15], common in semantic web services: Exact(Sim = 1), Plugin (2/3 <= Sim <1), Subsume (1/3 <= Sim < 2/3), Intersection (0 < Sim < 1/3), and Disjoint (Sim = 0). Computation of semantic and structural similarity between parameter names is done using the Java WordNet Similarity Library (JWSL) [17]. Due to the limited paper space, we show in Table 4 the obtained similarity values between Op11a of the service S11 and operations of the services that reply to `CityToZipCode` task.

3.2 Experiments Results

We use the previously obtained data in building an RCF, as described in Section 2. Thus, our non-relational contexts are composed of the obtained services together with their QoS values, illustrated in Table 3. They are $S_{1i} * QoS$, $S_{2j} * QoS$, and $S_{3k} * QoS$. The services inter-relation (relational) contexts are

[6] The WSDL URLs are available on: http://www.lirmm.fr/∼azmeh/wise10/expe/wsdls.html

Tasks	WSs names & identifiers		Operations names & identifiers		# Inputs	# Outputs
IPToCity	IP	S11	GetCityNameByIP	Op11a	1	1
	DOTSGeoPinPoint	S12	GetLocationByIP	Op12a	2	9
			GetCountryByIP	Op12b	2	6
			GetLocationByIP_V2	Op12c	2	12
	WsIp	S13	GetLocation	Op13a	1	3
	GeoCoder	S14	IPAddressLookup	Op14a	1	10
			PhysicalAddressLookup	Op14b	1	10
	XigniteHelp	S15	WhatsMyIP	Op15a	0	33
CityToZipCode	GeoPlaces	S21	GetPlaceDetails	Op21a	2	12
	MediCareSupplier	S22	GetSupplierByCity	Op22a	1	14
	DOTSGeoCoder	S23	GetGeoLocationByCityState	Op23a	3	5
			GetGeoLocationWorldWide	Op23b	4	5
	ClientScriptServices	S24	GetZipCodeCompletionList	Op24a	3	5
	FedAch	S25	GetAchByName	Op25a	3	5
	DOTSAddressValidate	S26	ValidateCityStateZip	Op26a	4	4
		S26	ValidateAddressSingleLine	Op26b	2	19
	ProMilesWebService	S27	GeoCodeLocation	Op27a	6	6
			GetZipsInRadius	Op27b	2	4
			ValidateLocation	Op27c	4	4
	CityZip_Service	S28	GetZipCode	Op28a	4	3
	USZip	S29	GetinfoByCity	Op29a	1	1
			GetInfoByState	Op29b	1	1
	DotSGeoCash	S210	GetATMLocationsByCityState	Op210a	2	8
	LPWebService	S211	GetLocationCities	Op211a	2	1
			GetLocationStates	Op211b	1	1
			GetLocationByCity	Op211c	3	1
	GeoServiceWebService	S212	GetZipCodesFromCityState	Op212a	3	1
ZipCodeToWeather	DOTSFastWeather	S31	GetWeatherByZip	Op31a	2	18
			GetWarningsByZip	Op31b	2	11
			GetFiveDayForecastByZip	Op31c	2	8
			GetWeatherHistoricalByZip	Op31c	4	18
	USWeather	S32	GetWeatherReport	Op32a	1	1
	WeatherForecast	S33	GetWeatherByZipCode	Op33a	1	13
	WeatherService	S34	GetWeatherData	Op34a	1	15

Table 2. WSs used in our experiments.

constructed using the composability values in Table 4. Having two composable services, means that the first service contains at least one operation that can be composed with one or more operations from the second one. The maximal level of composability between the operations belonging to the two services, is used as the level of composability of the two services. For example, from the Table 4, we can see that the service s_{11} has two values of composability with service s_{29}, which are "Exact" and "Plugin". Accordingly, we consider the composability level between s_{11} and s_{29} to be "Exact". After building the RCF, the next step is to integrate queries, in order to specify the required QoS of the service corresponding to each task in the orchestration. We define the following queries: $query1$ demands services from $S1i$ having a "Low A"and a "High RT"; $query2$ demands services from $S2j$ having a "High A" and a "Medium RT"; $query3$ demands services from $S3k$ having a "High A" and a "Low RT". We also specify the minimal accepted composability level, we set it to be "Plugin". Thus, the relational contexts become one single context, which is the fusion of the "Exact" and "Plugin" contexts for the two sets of relational contexts (S_{1i},S_{2j}) and (S_{2j},S_{3k}). Executing RCA on the built RCF gives us the relational concept

Task	Services	Availability	Response time
IPToCity	IP	High	Very high
	DOTSGeoPinPoint	Medium	Low
	WsIp	High	Very high
	GeoCoder	High	High
	XigniteHelp	Low	Medium
CityToZipCode	GeoPlaces	High	Medium
	MediCareSupplier	Very low	High
	DOTSGeoCoder	High	Medium
	ClientScriptServices	High	High
	FedAch	Very low	Low
	DOTSAddressValidate	High	Medium
	ProMilesWebService	Medium	Medium
	CityZip_Service	High	Medium
	USZip	Very low	Low
	DotSGeoCash	Medium	Medium
	LPWebService	High	Medium
	GeoServiceWebService	High	Low
ZipCodeToWeather	DOTSFastWeather	High	Medium
	USWeather	Very low	Low
	WeatherForecast	Very Low	Low
	WeatherService	Medium	Medium

Table 3. QoS values of the WSs used in our experiments.

lattices (RCL) in Fig. 7, in which, we have composable services with a "Plugin" composability level (or "Exact"), and having the expected QoS levels.

By looking to $query3$ in $L3$, we notice that it asks for services having a "Medium RT" and "Medium A". While the original $query3$, requests a "High A" and a "Low RT", as indicated above. Since there were no matches for the original query, it switches to the next best level of QoS. The services returned by the queries are as follows: $query1$ returns s_{12} in $c3$, and s_{14} in $c5$ (lattice $L1$); $query2$ returns s_{23}, s_{26} in $c13$, s_{28} in $c21$, and s_{212} in $c8$ (lattice $L2$); $query3$ returns s_{31} in $c0$ and s_{34} in $c4$ (lattice $L3$).

In order to clarify the results obtained by the RCL, we show the composability values between the returned services in Table 5. We notice that, the services s_{12} and s_{14} have a "Plugin" composability with the services s_{23},s_{26}, and s_{28}. Thus, s_{12} and s_{14} must have the concepts $c13$ and $c21$ in their attributes sets. By regarding the lattice $L1$ in Fig. 7, we notice that $c3$ and $c5$ inherits $Compose(L2) : c13$ and $Compose(L2) : c21$ from $c2$. This means that s_{12} and s_{14} are composable with s_{23},s_{26}, and s_{28}, which is true. In the same way, we notice that s_{23},s_{26}, and s_{28} are composable with s_{31}, and s_{34} in lattice $L3$. On the other hand, we can also notice s_{212} in $L2$, which can be composed with s_{31}, and s_{34}, but can not be composed with neither s_{12} nor s_{14}. This explains the absence of $Compose(L2) : c8$ as an attribute for $c3$ and $c5$ in $L1$.

4 Related Work

The related work can be organized in three categories:

Web Service Composability and Substituability. Many works in the literature addressed the composability and substituability of web services. Ernst et al. [7] present a method based on syntactic descriptions of Web services. This

Operation	Operations achieving `CityToZipCode` task	Similarity values & inferences	
`Op11a`	`Op21a`	0.682	Plugin
	`Op22a`	1	Exact
	`Op23a`	0.802	Plugin
	`Op23b`	0.789	Plugin
	`Op24a`	0.291	Intersection
	`Op25a`	0.292	Intersection
	`Op26a`	0.699	Plugin
	`Op26b`	0.534	Subsume
	`Op27a`	0.460	Subsume
	`Op27b`	0.643	Subsume
	`Op27c`	0.424	Subsume
	`Op28a`	0.766	Plugin
	`Op29a`	1	Exact
	`Op29b`	0.710	Plugin
	`Op210a`	0.903	Plugin
	`Op211a`	0.599	Subsume
	`Op211b`	0.377	Subsume
	`Op211c`	0.772	Plugin
	`Op212a`	0.614	Subsume

Table 4. Similarity values and inferences between `Op11a` and operations achieving `CityToZipCode` task.

	s_{23}	s_{26}	s_{28}	s_{212}
s_{12}	Partial	Partial	Partial	Subsume
s_{14}	Partial	Partial	Partial	Subsume

	s_{31}	s_{34}
s_{23}	Partial	Subsume
s_{26}	Partial	Intersection
s_{28}	Partial	Partial
s_{212}	Exact	Partial

Table 5. Composability values between the services of *query*1 and *query*2, then services of *query*2 and *query*3.

approach analyzes multiple invocations of these services. The input and output parameter values are compared and matchings are deduced. Contrarily to our approach, this work is based on the experimental usage of Web services, and is perfectly complementary to ours. In this research area, much work has been done on semantic web services [21,13,12,11]. In [13] and [12], the authors present a model for checking the composability of semantic web services at different levels: syntactic, semantic and quality of service. They first proposed a model for describing the static (non-computational) and dynamic (business) semantics of web services. Then, they formalized a set of rules that allow the computation of a composability degree.This work deals with two kinds of composability: horizontal (normal composability) and vertical (substituability). In our approach, web service semantics are extracted from service's signatures and documentation. We use tools (based on WordNet) to compute the similarity between names or documentation keywords of different web services. This allowed us to deduce the composaibility of services. We do not use a high-level model of web service semantics as in [12]. In addition, in our work we experimented the approach on real-world (not semantic) web services obtained from *Service-Finder*. The authors of [12] made however a simulation-based experimental study. In [13], the same authors deal with three quality attributes: fees, security and privacy. In our approach, we focus on other web service qualities, which are availabil-

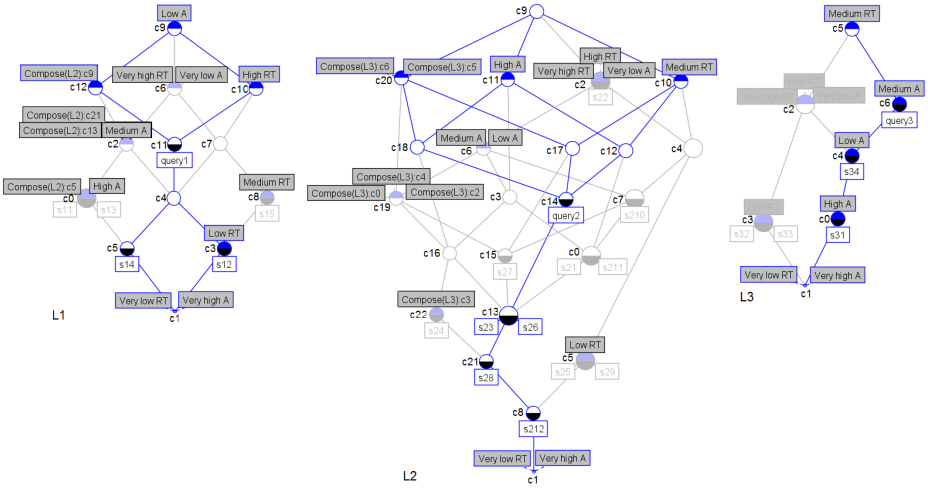


Fig. 7. The RCL corresponding to the services listed in Table 2.

ity and efficiency (response time). In [11], the authors present a method which combines semantic and static analysis of messages in semantic Web services. Data types (XML schemas) and semantic description (domain ontologies and SA-WSDL [20] specifications) of parameters (parts of web service messages) are used to deduce mappings. These mappings are then transformed into adapters (XSL documents). In our work, we concentrated on the selection of services which offer maximal QoS and which require, when composed, minimal adaptations. The two approaches are complementary.

Web Service Selection wrt QoS. In [24], Zeng *et al.* present a middleware platform that enables the quality-driven composition of web services. In this platform, the QoS is evaluated by means of an extensible multidimensional model, and the selection of web services is performed in such a way as to optimize the composite services QoS given a set of constraints and a set of candidate web services. Two service selection approaches for constructing composite services have been proposed: local optimization and global planning. Their study shows that global planning is better than local optimization. Aggarwal *et al.* [1] present a constraint driven web services composition tool that enables the selection and the composition of web services considering QoS constraints. Like Zeng *et al.* [24], a linear integer programming approach is proposed for solving the optimization problem. In [23], Yu *et al.* propose heuristic algorithms to find a near-to-optimal solution more efficiently than exact solutions. Yu *et al.* models the QoS-based service selection and composition problem in two ways: the combinatorial model defines the problem as a multi-dimension multi-choice knapsack problem and the graph model defines the problem as a multi-constrained optimal path problem. A heuristic algorithm is introduced for each model. Despite the significant improvement of these algorithms compared to exact solutions [24] [1], both algo-

rithms are not suitable for large service-based systems. This type of systems (i.e., composed of a large number of services) may be composed easily and efficiently using our approach.

Web Service Selection using Concept Lattices. Many works in the literature have addressed the classification of web services using concept lattices [3,4,2,16,6]. In [3], Azmeh *et al.* presented a tool which helps in classifying directories of web services from Seekda into concepts lattices. The classification is built upon contexts which associate services to operation signatures. In [4], the used contexts formalize relations between services and keywords extracted from their documentation, and similarity between operations. In these works, the classification is not based on QoS (filtering in [3] is done on only one quality attribute, availability). In addition, these approaches are based on FCA, and not on RCA, and don't base the classification on an orchestration. In [6], the authors present an approach where formal contexts associate web services and functionalities provided by these services as well as their non-functional properties (security). This interesting work addresses a different kind of quality attributes (security) that is critical in pervasive computing. Besides, it does not deal with service composition, and uses FCA (not RCA).

5 Conclusion

We presented an approach for assisting the development of a web-service orchestration based on a workflow including interacting abstract tasks, and on ready-made services from public registries. Relational Concept Analysis (RCA) helped us to take into account both QoS requirements and levels of composability in classifying candidate web services for implementing the tasks. The result is a set of web service lattices (one for each task), where the designer can see potentially collaborating services, as well as their QoS values. Inside each lattice, services are ordered by QoS and composability values. We have also proposed a querying mechanism that, given an expected level of QoS and composability, guides the designer in the lattices and highlights parts of the lattices where services should be chosen. We led an experiment on a sequential orchestration composed of three tasks. This illustrated and showed the feasibility of the approach.

As a future work, we plan to lead other experiments and to study other encodings of the problem with RCA, to give complementary views. In addition, we intend to study more deeply other composability computation techniques (based on other semantic descriptions of web services). This will allow us to obtain more precise compositions.

References

1. Aggarwal, R., Verma, K., Miller, J., Milnor, W.: Constraint driven web service composition in meteor-s. In: SCC '04. pp. 23–30 (2004)
2. Aversano, L., Bruno, M., Canfora, G., Di Penta, M., Distante, D.: Using concept lattices to support service selection. Int. J. of Web Serv. Res. 3(4), 32–51 (2006)

3. Azmeh, Z., Huchard, M., Tibermacine, C., Urtado, C., Vauttier, S.: Wspab: A tool for automatic classification and selection of web services using formal concept analysis. In: Proc. of ECOWS'08 (2008)
4. Azmeh, Z., Huchard, M., Tibermacine, C., Urtado, C., Vauttier, S.: Using concept lattices to support web service compositions with backup services. In: Proc. of ICIW'10 (2010)
5. Chambers, J.M., Clevelmd, W.S., Kleiner, B., Tukey, P.A.: Graphical methods for data analysis. Wadsworth & Brooks / Cole (1983)
6. Chollet, S., Lestideau, V., Lalanda, P., Colomb, P., Moreno, D.: Heterogeneous service selection based on formal concept analysis. In: Proc of Int. W. on Net-Centric Service Enterprises: Theory and Application (NCSE2010) (2010)
7. Ernst, M.D., Lencevicius, R., Perkins, J.H.: Detection of web service substitutability and composability. In: Int. W. on Web Services — Modeling and Testing (WS-MaTe 2006). pp. 123–135 (2006)
8. Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer Verlag (1999)
9. Huchard, M., Hacene, M.R., Roume, C., Valtchev, P.: Relational concept discovery in structured datasets. Ann. Math. Artif. Intell. 49(1-4), 39–76 (2007)
10. Huhns, M.N., Singh, M.P.: Service-oriented computing: Key concepts and principles. IEEE Internet Computing 9(1), 75–81 (2005)
11. Lécue, F., Salibi, S., Bron, P., Moreau, A.: Semantic and syntactic data flow in web service composition. In: Proc. of the ICWS'08. pp. 211–218 (2008)
12. Medjahed, B., Bouguettaya, A.: A multilevel composability model for semantic web services. IEEE Trans. on Knowledge and Data Eng. 17(7), 954–968 (2005)
13. Medjahed, B., Bouguettaya, A., Elmagarmid, A.K.: Composing web services on the semantic web. The VLDB Journal 12, 2003 (2003)
14. Menascé, D.A.: Qos issues in web services. IEEE Internet Comp. 6(6), 72–75 (2002)
15. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.P.: Semantic matching of web services capabilities. In: In proc. of ISWC '02. pp. 333–347 (2002)
16. Peng, D., Huang, S., Wang, X., Zhou, A.: Management and retrieval of web services based on formal concept analysis. In: Proc. of CIT'05. pp. 269–275 (2005)
17. Pirró, G., Seco, N.: Design, implementation and evaluation of a new semantic similarity metric combining features and intrinsic information content. In: ODBASE 2008. pp. 1271–1288 (2008)
18. Rao, J., Su, X.: A Survey of Automated Web Service Composition Methods. LNCS, Springer, Heidelberg (2005)
19. Service-Finder: `http://www.service-finder.eu/`
20. Sivashanmugam, K., Verma, K., Sheth, A., Miller, J.: Adding semantics to web service standards. In: Proc. of ICWS'03. pp. 395–401 (2003)
21. Sycara, K., Paolucci, M., Ankolekar, A., Srinivasan, N.: Automated discovery, interaction and composition of semantic web services. Journal of Web Semantics, Elsevier 1, 27–46 (2003)
22. Valle, E.D., Cerizza, D., Celino, I., Lausen, H., Steinmetz, N., Erdmann, M., Schoch, W., Funk, A.: Realizing service-finder, web service discovery at web scale. In: ESTC '08 (2008)
23. Yu, T., Zhang, Y., Lin, K.J.: Efficient algorithms for web services selection with end-to-end qos constraints. ACM Transactions on the Web 1(1), 6 (2007)
24. Zeng, L., Benatallah, B., Ngu, A.H., Dumas, M., Kalagnanam, J., Chang, H.: Qos-aware middleware for web services composition. IEEE Transactions on Software Engineering 30(5), 311–327 (2004)